Design Document: Essay Scoring Engine

1. Introduction

S-Space (http://code.google.com/p/airhead-research/) is a semantic space package that implements various algorithms e.g. Latent Semantic Analysis (LSA), Random Projection. A full list of all implemented algorithms may be found here (http://code.google.com/p/airhead-research/wiki/SemanticSpaceClasses).

The number of words in an essay may be of large order and it will be important to identify key characteristics (e.g. words, word-word relations) that may be thrown into a regression analysis. This is where semantic spaces algorithms come into play. Semantic space algorithms depict semantics of a word in a context and hence allow reduction in "dimensions". Dimensions may be simply the number of word-word relationships or unique terms. In this document I will focus on the use of LSA for the purposes of Essay Scoring.

Before proceeding further it is worth noting that many of these have parallels in Machine Learning: Principal Component Analysis (http://en.wikipedia.org/wiki/Principal_component_analysis) and differ primarily in the mathematics used.

Results are in Appendix A and all of the following discourse may be skipped except for the diagram in section 5.

2. Dimension Reduction using S-Space(LSA)

The full details of LSA are beyond the scope of this document. Please refer to [3]. However, key steps for the sample run for preparing this document are outlined below. For each of essay sets 1, 3, 4 and 5 in Appendix A the following steps were followed.

Inputs:

1. Number of desired dimensions k = 50. *This should be a variable based on word length of the essay.*
2. Training documents (Taken from Kaggle competition: http://www.kaggle.com/c/asap-aes/download/training_set_rel3.xls). For each essay set there were about 1700 training essays and about 20 that I used for validation of the models.

Steps:

1. Build a term vector for each document. Each term is a unique word appearing in the document.
2. Put all these term document vectors into a single matrix.
3. Perform SVD into matrices U, V and W^T.
4. V is a diagonal matrix. Pick the largest k values from V into V' and readjust matrices, U and W into U' and W'.
5. Calculate new term-document matrix: U' V' W'^T.

Note: The description of these new term-spaces is unnecessary for our purposes.

3. Use of S-Space in Essay Scoring: How To

Step 1: Training / Model Building

1.1 For every training document clean up the document and tokenize. Please see section 4.
1.2 Run the training documents through LSA to reduce dimensions.
1.3 Note down matrix U, V and W (Singular Value Decomposition of the term-document matrix into UVW^T) - needed for step 2.3 below.
1.4 Note down the original term vector – needed for step 2.2 below.
1.5 Use the new reduced dimension term-document vectors along with any other parameters in a regression algorithm along with the human assigned scores.
1.6 Note down the model.

Step 2: Scoring an essay.

2.1 Follow the same cleaning and tokenizing procedure as in step 1.1.
2.2 Construct the term vector for the response based on original terms as noted in step 1.4. – let's call this d.
2.3 Calculate V^{-1} and W^T for the vectors V and W as noted in step 1.3.
2.4 Calculate the term vector in the reduced dimension using the formula $V^{-1} W^T$ d.

2.5 Use the term vector calculated above along with the model trained in step 1.6 to calculate the final score.

Step 1.1, 1.3 and 1.4 have to be implemented as they are not there in the baseline implementation of S-Space. Step 2 is the core of essay scoring and needs to be implemented.

4. Use of S-Space in Essay Scoring: Pit Falls

There are a few pit falls of using the baseline implementation:

1. Custom tokenization has to be implemented. The baseline implementation catches noise words e.g. @CAPS11, @CAPS8.
2. It is catching too many unique words e.g. episode, episodes.
    a. Stemming,
    b. Stopwords.
3. Spelling mistakes are a problem: engineers, enginere.
4. Possibly use TF/IDF (see section 5.2).

5. Algorithm Flow

Figure 1 diagrammatically depicts the steps in the training and scoring processes. Steps 1, 2, 3 and 4 are common during the course of training and scoring. Below is a brief description of the various steps. In section 5.1 we will list the various libraries we use to accomplish these steps.

1. Tokenization: The response may not be well formed e.g. "…end of sentence one.beginning of sentence two…". A crucial first step will be to tokenize so as to make our task easier during parts-of-speech (POS) tagging. This step may also tokenize seemingly compound words e.g. converting "this/that" into "this", "/" and "that".

2. Sentence Detection: Sentence detection is the first step in building a parse tree and eventual POS tagging.

3. POS tagging: Identifying parts-of-speech are crucial for reducing dimensions for two reasons. First, this allows us to reduce various forms of a word to a canonical form e.g. "relatives", "relative", "related" to "relate" – done in step 4. The second reason this may be important is in judging features in the response to be used as additional parameters in the scoring process.

4. Stemming/Base word identification: explained above with the addition of spelling. Words may be spelled incorrectly and it may be required to reduce them to a correct canonical form.

5. Build term-by-document matrix: Term-by-document matrix relates the weight of a term in each training document. The weight may simply be the frequency of that particular word in respective documents. More complex weight calculations are also possible e.g. if TF/IDF is available for a large dataset then that may be used to assign more appropriate weights which will automatically take care of common words/stop words. More is discussed in section 5.2.

A sample term-by-document matrix is shown in the table below:

Terms	**doc1**	**doc2**	**doc3**	**doc4**
x	1	0	3	0
y	2	1	1	3
z	0	4	1	4

Table1: A sample term-by-document matrix. So the document vector for term x is (1, 0, 3, 0) and the term vector for document 1 is (1, 2, 0).

One of the by-products of this step is a record of term positions in the document vector. So for example in the above sample term x occupies the first value in the document vector, term y occupies the second and so on. This will be required in later steps.

Also see section 5.2 for the alternate approach to assigning term weights.

6. Semantic Spaces Calculations: Use LSA to calculate the semantic spaces. One of the crucial parameters required for this step is the input on how many dimensions are desired. This will have incidental outputs e.g. the semantic spaces and any other data that are used in calculations e.g. the Singular Value Decomposition (SVD) matrices for the term-by-document matrix when using Latent Semantic Analysis (LSA).

7. Regression Analysis: Use the reduced dimension semantic space document vectors along with human assigned scores in the regression analysis to calculate scoring models. These scoring models need to be stored for later use during scoring.

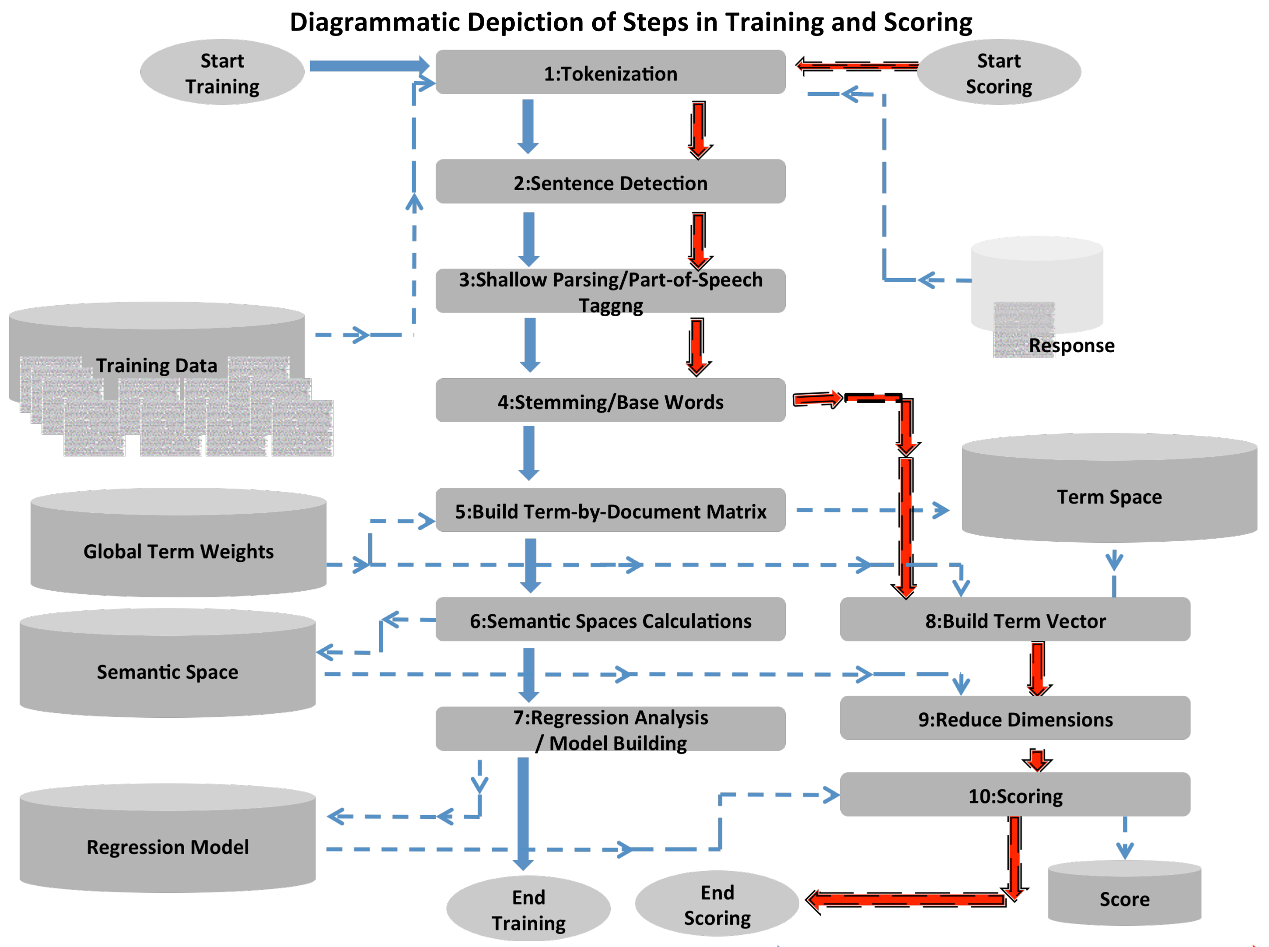

Figure 1: Diagrammatic representation of steps in the training and scoring process. ➡ indicates flow in the training process. ⇨ indicates flow in the scoring processes. The dashed lines indicate data flow.

8. Build Term Vector: The original training documents had been reduced to a different semantic space during step 6 and the representation in that reduced dimension semantic space had been used in regression analysis for model building in step 7. To score a new response, it thus means, that the first step has to be to represent that document in the new reduced dimension semantic space. The mathematics of that have been noted down in section 2, step 2. We will use the dimension information for terms as noted down in step 5 above to first build a term vector in the original dimension.

9. Use the term vector derived in step 8 above along with semantic spaces derived in step 6 to reduce the query document/response to the lower-dimensional semantic space.

10. Plug in the values for the various dimensions as derived in the regression analysis (step 7) to calculate a final score.

5.1 Packages to be used

Below are lists of front runner open source libraries for use in the essay scoring engine.

OpenNLP: Supports tokenization, parts-of-speech tagging, sentence detection and chunking (i.e.shallow parsing where only the type of the word is identified. For example in shallow parsing the sentence "Shaw Publishing offered Mr. Smith a reimbursement last March." Will be parsed as "[$_{\text{AGENT}}$Shaw Publishing] **offered** [$_{\text{RECEPIENT}}$Mr. Smith] [$_{\text{THEME}}$a reimbursement] [$_{\text{TIME}}$last March]." OpenNLP also provides APIs for parts-of-speech tagging. OpenNLP is intended for us in steps 1, 2 and 3 as noted above.

The caveat so far is that that it seems that OpenNLP as implemented in Java language may have problems with multi-threading. If this becomes an issue one of the alternated packages for natural language processing as listed below will be used.

Alternate packages are GATE, Stanford Core NLP (CoreNLP).

Java WordNet Library (JWNL): WordNet is used for root word identification in step 4. JWNL seems fairly up-to-date. Alternate packages are Java API for WordNet Searching(JAWS), Java WordNet Interface (JWI). JWNL seems to have been the most recent.

S-Space: S-Space is used as a library for semantic space calculations. It is highly customizable i.e. custom implementations may be plugged in for various algorithms. We see ourselves taking advantage of this for our purposes as has been done at a more coarse level in the prototype essay scoring engine.

Semantic Vectors is an alternate possibility.

Apache Commons Math Library: Used for various statistical tasks e.g. regression analysis. This seems to be fairly active and up-to-date. jHelpWork seems to be another package that may be of use for statistical purposes. It also has neural network training models. On the surface of things, it does seem that jHelpWork internally uses an older version of Apache Commons Math for statistical computations.

Lucene: Internally S-Space does use Lucene. Lucene may have some uses in calculating Term Frequency (TF)/ Inverse Document Frequency(IDF). TF/IDF may have a use in calculating better weights for use in term-by-document matrix. The difficulty with using TF/IDF for our purpose is that we may not have a good way of calculating global values. Running the TF/IDF calculation algorithms on the training response set may be a close approximation however. More on this in section 5.2 below.

5.2 Alternate approaches to assigning term weights in step 5.

In step 5 of section 5 we assign term weights to the term-by-document matrix. Our approach so far has been simple: assign the frequency for a particular term as the weight for that term in the matrix. This has a severe pitfall: use of unique words is not given any higher weightage than use of common words. So, in the absence of appropriate stop words algorithms common words e.g. "if", "the" may end up with large values in SVD as used in LSA.

One option is to use Term Frequency(TF)/Inverse Document Frequency(IDF) values instead. TF/IDF gives weightage to terms based on their global uniqueness as well as frequency in a document. Unique words will get somewhat higher weightage. These metrics are really useful in search where a document that has a unique term matching a query will end up getting scored higher.

For our purposes we do not have a way to get their global values as our document space would be quiet small: limited to the training set. The training set however, may be a good approximation. So we will calculate the TF/IDF scores for each term in the training set and use that in the term-by-document matrix. So during the training set we will calculate the IDF for each term t in the full document space D(IDF(t, D)) and TF for each term t in each document d in the training set i.e. D (TF(t, d)). We will need to preserve the IDF(t,D) values for use in scoring.

So here are the steps.

During Training
Step 1: Calculate IDF(t,D) for every term in the full document space D. as well as TF(t, d) where D is the full training document space.
Step 2: Simultaneously calculate TF(t,d) for every term t for every document d in the training space.
Step 3: Use the TF(t,d) * IDF (t, D) in the term-by-document matrix during training.

During Scoring
Step 1: Calculate the TF(t,r) values for every term t in the response r.
Step 2: Calculate the TF(t, r) * IDF(t,D) for every term t in the response r and derive the term vector for the response r.
Step 3: Use this term vector for r in step 9.

This has not been implemented in the prototype code but will be implemented in the final version – the implementation will make use of Lucene. It will be configurable so that the user may select not to use it.

5.3 Incremental vs. Full semantic Spaces.

Some research [1] has shown that as few as 300 training essays may be sufficient to derive a good model. This still begs the question of what would be considered a representative sample and how to proceed as new responses show up later significantly scaling up the number of unique words being seen.

This problem impacts steps 3, 5, 6, 7 and 8 of section 5 above. There are two possible options: 1) Increase data points 2) increase the number of dimensions/factors.

For large essays the time taken to do POS/shallow parsing (step 3) is significant but still sub-second and the total time taken to run this step for all the available documents in the training set would scale linearly. So this step should not be a problem in terms of time take. In terms of data variability with increasing sample space also, this step should not be a problem.

Steps 5, 6 and 8 are most susceptible to variation in the terms spaces. Unfortunately there is no simple solution but to run the training all over again. A good indicator of this will be how many new terms are being encountered during step 8. The implementation will keep track of this metrics.

The time take for regression analysis (step 7) for 1700 data points with 50 factors is reasonably fast – finishing within as little as a couple of minutes. At the time of writing this design document it is not clear as to how the time taken for regression analysis scales with number of data points and the number of dimensions.

6. Bibliography

[1] Sargur Srihari, Jim Collins, Rohini Srihari, Harish Srinivasan, Shravya Shetty, and Janina Brutt-Griffler. "Automatic Scoring of Short Handwritten Essays in Reading Comprehension Tests."
[2] Keith Stevens, David Jurgens. "The S-Space Package: An Open Source Package for Word Space Models."
[3] Thomas K Landauer, Peter W. Foltz, Darrell Laham. "An Introduction to Latent Semantic Analysis."